Take Me to SSD: A Hybrid Block-Selection Method on HDFS based on Storage Type

Minkyung Kim
Yonsei University
50 Yonsei-ro, Seodaemun-gu
Seoul, Korea
+82 2 2123 7757
goodgail@cs.yonsei.ac.kr

Mincheol Shin
Yonsei University
50 Yonsei-ro, Seodaemun-gu
Seoul, Korea
+82 2 2123 7757
smanioso@cs.yonsei.ac.kr

Sanghyun Park[1]
Yonsei University
50 Yonsei-ro, Seodaemun-gu
Seoul, Korea
+82 2 2123 7757
sanghyun@cs.yonsei.ac.kr

ABSTRACT
As the era of Big-data has risen, the importance of big data technologies is also increasing day by day. Especially, Hadoop has become a critical part of the overall Big-data system because of its ability to store, process, and analyze thousands of terabytes of data. A major issue for supporting high performance on Hadoop is managing the growth of data while satisfying high storage I/O request. Hadoop's overall performance is largely influenced by the storage input/output(I/O). However, storage I/O technologies are still very limited. Therefore, now more than ever, studies on improving storage I/O on a distributed file system of Hadoop(HDFS) have been gaining popularity. To this end, latest trend in storage systems is to utilize hybrid storage devices. However, it is not easy to use the information of heterogeneous storage devices in HDFS. This is because, when reading data, HDFS is unable to exploit such heterogeneous storage type information yet.

In this paper, we propose a hybrid block-selection method on the HDFS, we consider the storage type such as SSD and HDD when reading data. Using this method, the Hadoop Eco System utilizes the high SSD bandwidth by priority. As a result, we certainly improve the Hadoop Eco System overall performance. In the experiments, we demonstrated that our new method efficiently reduced the execution time of select count(*) query and TPCH benchmark up to 22% and 30% on average.

[1] Corresponding author. Tel: +82-2-2123-5714; Fax: +82-2-365-2579.; E-mail address: sanghyun@cs.yonsei.ac.kr (S.Park).

SAC 2016, April 04-08, 2016, Pisa, Italy
© 2016 ACM. ISBN 978-1-4503-3739-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851658

CCS Concepts
• Information system → Database management system engines

Keywords
Hadoop, SQL-on-Hadoop, SSD, HDD, Storage I/O

1. INTRODUCTION
Big-data is an evolving term for extremely large or complex data sets. It produces terabyte or petabyte quantities of data in a short period of time. As the data amount is increased, the need for big data analysis to extract meaningful information from a huge amount of data has increased significantly. However, traditional relational database management systems (RDMS) are inadequate for very-large data sets storage and processing. Consequently, studies on data store, access, manipulation and analysis specializing in Big-data have been actively carried out.

There are various applications that allow handling and processing of Big-Data, the basic framework has been that of Apache Hadoop. Hadoop comprises of two parts, which are the data storage part(HDFS) and the other being the data processing part(MapReduce). For data processing, MapReduce divides tasks and allows their execution in parallel. MapReduce is limited in that it is not suitable for real-time processing and does not support the SQL language. To this end, different methods of processing Big-Data, such as SQL-on-Hadoop, have been developed. Over the years various attempts have been made to improve Big-Data's real time processing performance on SQL-on-Hadoop with the combination of HDFS.

HDFS is a distributed file system of Hadoop. HDFS divide Big-data into small unit of data. The small data are replicated, distributed and stored in a large number of storage devices. Recently, HDFS optimization techniques have been proposed using SSDs with high I/O performance[1,8,13,16,19]. However, it is still difficult to replace all existing HDD devices with SSD devices. New solutions, that simultaneously use both SSDs and HDDs, are being suggested to solve the problem.

In this paper, we suggest a hybrid block selection method on HDFS based on storage type. We consider the type of storage device and select optimal location to read block. If the same block exists in both HDD and SSD among different candidate

DataNodes, the SSD with relatively higher performance is desired in preference to the HDD. To accomplish our approach, we used HDFS which consists of mixing SSDs and HDDs. We defined and implemented a new algorithm on the SQL-on-Hadoop distributed processing system. Finally, we evaluated storage I/O performance and overall Hadoop system performance. Experiments show that the execution time of select count(*) query was improved up to 22% on average. TPCH benchmark results demonstrate that the execution time of queries reduced from 4% up to 30% on average. By changing block selection priority in the existing system without introducing any additional systems, we can achieve improvement of overall performance in Hadoop Eco Systems.

This paper has the following organization: Section2 explains background of our study. Section3 introduces related work. Section4 discusses methods. Section5 shows experimental configurations and results. Section 6 is the conclusion.

2. BACKGROUND

This section discusses background of our study. Subsection2.1 explains HDFS. Subsection2.2 explores SQL-on-Hadoop. Finally, Subsection2.3 describes the advantages and disadvantages of SSDs.

2.1 Hadoop

Hadoop[14] is open-source software that allows for the distributed processing of large data sets across cluster. Architecturally, Hadoop consists of a storage part (HDFS) and a processing part (MapReduce). MapReduce deals with data processing and the generation of large data sets with a parallel, distributed algorithm on a cluster. HDFS is a distributed file system. Its basic composition consists of a single NameNode and DataNodes with many slaves. Figure1 illustrates the overall architecture of the Hadoop system.

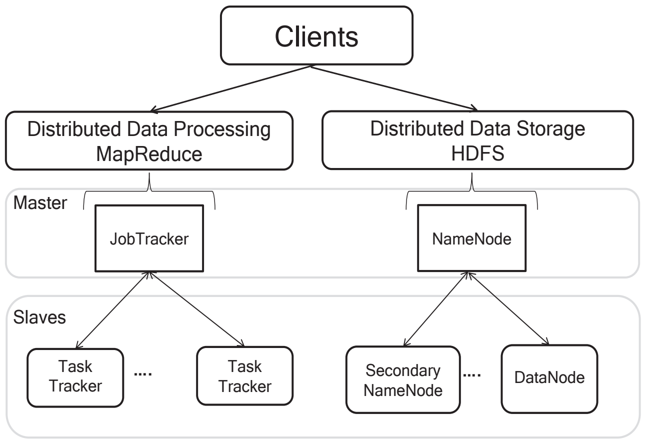

Figure 1: Overall architecture of the Hadoop system

The following process deals with Hadoop client requests in more detail. When the client's write request is delivered to the Hadoop system, HDFS divides a file into block units, and makes multiple copies of a block to support fault tolerance. If the default factor of block-replica is 3, each block has 3 replicas. Each replica is allocated on different DataNodes. After each block is written, its location is managed by NameNode. When a HDFS client such as distributed processing system read a file, a HDFS client requests the blocks-locations information and selects the appropriate blocks locations based on the received information from NameNode.

Conventional HDFS identifies several types of storage devices as only DISK type, without distinguishing the type of storage device. Therefore, the storage-type information is not stored. However, the recently released Hadoop supports an archival storage, SSD, and memory API. It enables us to recognize various storage device types, such as SSD, HDD, and RAM, using a heterogeneous framework. Based on the heterogeneous framework, the API offers six types of storage policies (ALL_SSD, ONE_SSD, etc). Each policy defines how to store blocks in SSD or HDD.

2.2 SQL-on-Hadoop

MapReduce is limited in that it is not suitable for real-time processing and does not support the SQL language. To solve this problem, different methods of processing Big-data have been developed[9,11,18]. A representative example is SQL-on-Hadoop [3]. It is a type of analytical application tool on the Hadoop platform; e.g., Hive [15], Tajo [2], Impala, Spark, etc.

There have been numerous attempts to improve real time processing performance on SQL-on-Hadoop with the combination of HDFS. For example, Tajo is used with HDFS. Tajo is a Big-data relational and distributed data-warehouse system. Tajo consists of a single master and a number of slave workers logically. One of the workers acts as a query master. A query master is in charge of managing a query process. As soon as client sends a query, Tajo starts to create a plan. A plan has a number of tasks which is the execution unit of Tajo. A query master schedules suitable workers for performing the process from among many candidate workers. The selected workers access the HDFS level to read/write and process HDFS blocks.

2.3 Solid-State drive(SSD)

SSD give large benefits over HDD. One of the advantages is that the SSD writes data not on magmatic disk but on semiconductor memory e.g., NAND flash memory. Accordingly, the long search and delay time followed by a conventional HDD header and flutter can be largely reduced. Therefore, SSDs have high sequential, random I/O performance than HDDs. Another advantage is that the SSDs have high energy efficiency. It consumes less power at the same point than HDDs[10].

The disadvantages of SSDs are much more expensive than the HDDs until now. The expensive cost causes SSDs difficult to become popular, even though it had great performance. However, recently, thanks to the oversupply of NAND flash memory, the price of SSDs is now much cheaper than before. Cost-per-bit of SSDs continues to decrease compared to the past. Therefore, in the future, utilizing SSDs technology is expected.

3. RELATED WORK

In this section, we present a summary of introducing SSDs to the Hadoop system that have been proposed by previous research. Sangwhan, Jaehwan and Yang-suk [10] proposed three types of MapReduce performance: only SSDs, SSDs with HDDs, and only HDDs. The best storage throughput is the only-SSDs case.

However, proposed paper indicates that replacing all HDFS HDDs with SSDs is appropriate when it is imperative to increase performance despite obvious significant cost differentials. Consequently, mixing HDDs and SSDs within a Hadoop system has been considered.[4,5,7,12,17] Kambatla and Chen [6] also demonstrated the experiments result of comparing SSD performance to HDD performance within a Hadoop MapReduce framework. The experiment results demonstrate that SSDs deliver up to 70 percent higher MapReduce performance compared to HDDs of equal aggregate I/O bandwidth. It is also possible to predict high I/O performance when mixing SSDs and HDDs through the results of existing research.

However, researches based on selecting HDFS's blocks using the storage type are actually insufficient. Until now, HDFS clients can define where and how to store blocks in different types of storage devices. But, when HDFS clients read blocks, they usually do not consider the storage type of device. Therefore, blocks on SSDs or HDDs are read randomly. There is some scope to improve block selection techniques for storage I/O performance.

4. METHOD

Our new block selection method was applied to SQL-on-Hadoop with the combination of HDFS. HDFS is a commonly used file system in the Big-data field. We use Tajo which is one of the SQL-on-Hadoop systems.

4.1 HDFS

The physical storage devices in the HDFS were constructed as a mixing of SSDs and HDDs, based on the contents presented in [Subsection 3]. The HDFS default block-replica factor is 3. We used the ONE_SSD mode of archival storage, SSD, and memory API[Subsection 2.1]. Therefore, when block replicas are written, one replica on an SSD and the others on HDDs. Figure 2 describes the result of writing a Block B_0 to HDFS.

A heterogeneous framework of Hadoop enables us to recognize various storage device types, such as SSD, and HDD. In our investigation, the information of the storage type is not yet stored in BlockLocation class. In this paper, in order to take advantage of the high SSD I/O performance, we add the storage type of each block location as a member variable of the BlockLocation class.

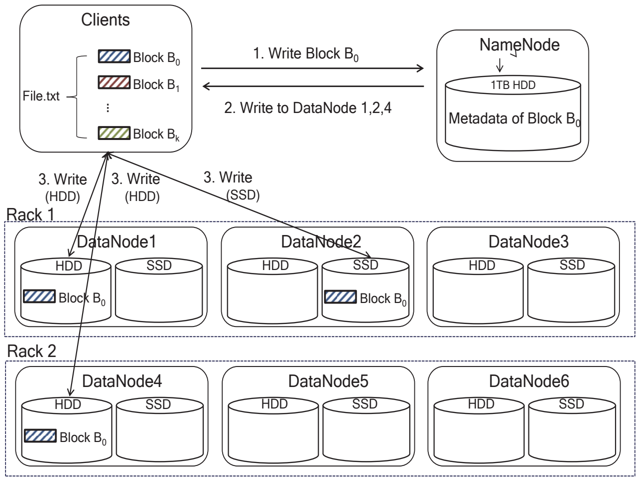

Figure 2: Writing Block B_0 to HDFS

4.2 Hybrid HDFS Block-Selection Method

As soon as a HDFS client sends a Block B_0 read requests, NameNode sends BlockLocation information to the a HDFS client. The BlockLocation information includes a list of DataNodes which have Block B_0 replicas. DataNodes in the list are sorted in the order of a short distance from the client. Figure 3 shows the general read process of a Block B_0. The general approach selects a Block B_0 replica located in the first of the DataNodes list. Therefore, in Figure 3, the HDFS client selects DataNode1 by priority. A Block B_0 replica located in the HDD of DataNode1 is read.

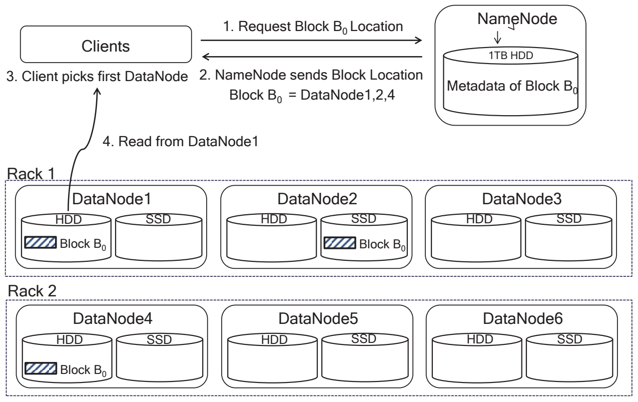

Figure 3: General method of reading Block B_0

The storage I/O performance depends on amounts of blocks read in the SSD. If a number of blocks in the HDD are more read than SSD, the storage I/O performance will be lower. To address this problem, we suggest the hybrid block-selection method on HDFS based on storage type. We use the modified HDFS BlockLocation class presented in [Subsection 4.1], which includes storage-type information. When a HDFS client chooses a Block B_0 replica among different candidate DataNodes, if B_0 replica in the SSD is available, it reads that block by priority. However, if B_0 replica in the SSD is not available, then it reads B_0 in the HDD.

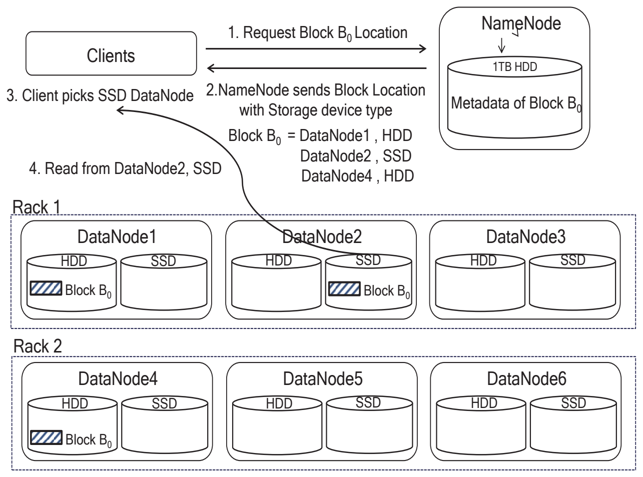
Clients
1. Request Block B_0 Location
NameNode
1TB HDD
Metadata of Block B_0
3. Client picks SSD DataNode
2.NameNode sends Block Location with Storage device type
Block B_0 = DataNode1 , HDD
DataNode2 , SSD
DataNode4 , HDD
4. Read from DataNode2, SSD
Rack 1
DataNode1
DataNode2
DataNode3
HDD
SSD
Block B_0
Rack 2
DataNode4
DataNode5
DataNode6

Figure 4: Our approach to reading Block B_0

Figure 4 shows our approach. Block B_0 is stored in DataNodes1, 2, and 4. Based on our approach, the HDFS client such as SQL-on-Hadoop checks the storage type of candidate DataNodes and selects DataNode2 by priority compared to other candidate DataNodes. This is because the Block B_0 is located in the SSD.

The proposed method was applied to Tajo. Tasks of Tajo has two types. One is LeafTask and another is NonleafTask. Scan tasks belong to the LeafTasks. NonleafTask includes tasks such as SORT, JOIN, etc. We focus on LeafTasks, especially scan tasks. If there is a scan task request, firstly, the scan task need to be registered in the LeafTasks. *Algorithm 1* shows the registration process of scan task to LeafTasks.

Algorithm 1 addLeafTask

```
    Description
    addLeafTask () is used for saving LeafTask information.
    e.g.  In case of Scantask, scantask's  information is added to
          storage devices on DataNode which has scantask's
          blocks.
    Input
    taskEvent : Task event that will be scheduled
    Output    none

    /* get information of event */
1   task=getTask (event)
    /* If event is scan task,  get block replica locations */
2   DataLocationList= getDataLocations (task)
3   for (DataLocation location : DataLocationList)
    /* DataNodeInfo means each location(DataNode) */
4   DataNodeInfo = location.getDataNodeInfo()

    /* get existing Mapping information of each storage device
    on current DataNode

    StorageTaskMapping consists of (key, value).
    key :  storage device ID  of  DataNode
    value : a list of tasks which is not yet performed
    */
5   StorageTaskMapping = StorageMapping(DataNodeInfo)
6   /* add scan task information to the task mapping of storage
    device */
    StorageTaskMapping.addTask(location.getStorageId(),
    location.getStorageType(),task)
7   End
```

In case of Tajo, a Tajo query master runs appropriate Tajo worker from candidate workers randomly. And then Tajo worker selects blocks to read. Therefore, we applied our new methods when Tajo worker selects blocks. The result of applying our method to the existing block selection method of Tajo, it has the following new block selection priority as follows:

➢ **New Priorities of Our Approach**

Priority 1. block in SSD of Local DataNode

Priority 2. block in HDD of Local DataNode

Priority 3. block in SSD of Same Rack DataNode

Priority 4. block in HDD of Same Rack DataNode

Priority 5. block in SSD of Random DataNode

Priority 6. block in HDD of Random DataNode

Algorithm 2 shows assignToLeafTasks method in more detail. It is used for selecting block replica based on the data location and storage type. In environments that run multiple workers at the same time, we can get the effectiveness that overall system preferentially use the performance of the SSD.

In case of Figure4, worker2 which is located with DataNode2 is used for executing LeafTasks. Based on our new priorities, a worker2 read a Block B_0 replica in the SSD of Local DataNode .Block B_0 are read completely, a scan task information which is related to Block B_0 is removed from LeafTasks. (Lines 4 to 7 of *Algorithm 2*). And then, worker2 resources is also used for executing other LeafTasks.

Algorithm 2 assignToLeafTasks

```
     Description
     A distributed processing system selects appropriate block
     to be read.
     Input
     taskRequests  : Candidate Tajo workers which have blocks
                     on Local DataNode
     LeafTasks    : Set of LeafTask
     LeafTaskHosts : List of DataNodes which have blocks
     Output    none

     /* scan tasks exist, taskRequests is also available */
1    while((LeafTasks.size() > 0 && !taskRequests.isEmpty())
      /* Select appropriate Tajo worker */
2     taskRequest = getTaskRequest (taskRequests)
      /* Save DataNode information of current Tajo worker */
3     DataNode = taskRequest.getDataNode()
      /* Priority 1. Local SSD blocks on current DataNode */
4     LocalTask = allocateLocalTask (DataNode)
5     if (LocalTask != null && LocalTask.getStorageType() ==
      "SSD")
6       executeTask (LocalTask)
7       LeafTasks.remove (LocalTask)
8     else
      /* Priority 2. Local HDD blocks on current DataNode */
9       LocalTask = allocateLocalTask (DataNode)
10      if(LocalTask != null)
11        executeTask (LocalTask)
12        LeafTasks.remove (LocalTask)
13      else /* LocalTask == null */
        /* Priority 3. Rack SSD blocks on current DataNode */
14      RackTask = allocateRackTask (DataNode)
15      if(RackTask != null && RackTask.getStorageType() ==
        "SSD")
16        executeTask (RackTask)
17        LeafTasks.remove (RackTask)
18      else
        /* Priority 4. Rack HDD blocks on current DataNode */
19      RackTask = allocateRackTask (DataNode)
20      if(RackTask != null)
```

```
21        executeTask (RackTask)
22        LeafTasks.remove (RackTask)
       /* Scan block still remaining */
       /* Priority 5. Random DataNode SSD task allocation */
23     if(LeafTasks.size() > 0)
24       RandomTask = LeafTask.DataLocation();
25       if(RandomTask != null &RandomTask.getStorageType()
             == "SSD")
26         executeTask (RandomTask)
27         LeafTasks.remove (RandomTask)
28       else
         /* Priority 6. Random DataNode HDD task allocation */
29       if(LeafTasks.size() > 0)
30         RandomTask = LeafTask.DataLocation()
31         if(RandomTask != null)
32           executeTask (RandomTask)
33           LeafTasks.remove (RandomTask)
34   End
```

As another example, if there remains scan tasks, a worker2 resources are used for reading remaining blocks e.g., Block B_1. For reading blocks efficiently, a worker2 selects appropriate block replica based on our new priorities. If there are no blocks replica in the SSD of Local DataNode, a worker2 find blocks replica in the HDD of Local DataNode. If there exists blocks replica, such as Block B_1, the worker2 read it. After completing a Block B_1 scan task, the information of a Block B_1 scan task is also removed from LeafTasks.(Lines 9 to 12) However, if there are no a Block B_1 in Local DataNode, the worker2 needs to read the block remotely. Tajo supports the function of reading blocks remotely. Worker2 finds a Block B_1 replica in the same rack server to minimize bandwidth consumption and read latency. If a Block B_1 replica exists in the SSD of same rack DataNode, a worker2 reads the block and removes a Block B_1 scan task from LeafTasks.(Lines 14 to 17) However, if there is no Block B_1 replica in SSD and the block is in the HDD, read a Block B_1 replica from HDD of the same rack DataNode. After finishing a Block B_1 scan task, remove it from LeafTasks. (Lines 19 to 22) Until now, if there is no Block B_1 replica in the same rack, worker2 gets the Block B_1 replica location directly (B_1 location is called Random DataNode). Then, a worker2 accesses the Random DataNode. If the Block B_1 replicas exist both SSD and HDD, it reads the block in the SSD first and removes information of the scan task from LeafTasks. (Lines 24 to 27) But, if a Block B_1 replica is in the HDD, read it from HDD and removes information of the scan task from LeafTasks. (Lines 30 to 33)

Our approach operates on HDFS. Therefore, a hybrid block-selection method can be also applied to other distributed processing systems. By simple storage-type checking without introducing any additional new system, Hadoop system can take advantage of the high SSD I/O bandwidth.

5. EXPERIMENTAL RESULTS

5.1 Experimental Settings

We conducted all experiments comparing original approach to hybrid block selection method in terms of execution time of SQL queries. We used 100G datasets. The datasets were generated by the TPC-H data generator. All experiments were conducted on a cluster that consists of four machines. Each machine had the same environment settings. The hardware and software settings of this paper are as follows.

- **Hardware Setting**

CPU	Intel 4 Core i7 CPU
Memory	32 GB memory
Disk	2TB HDD with OCZ Vector 120GB SSD

- **Software Setting**

OS	Linux OS, kernel version 3.10
File System	extended file system 4 (ext4)
Software	Hadoop 2.7.1, Tajo 0.10.1

One machine is used for a NameNode of HDFS and a master of Tajo. Three machines have a DataNode of HDFS and a Tajo worker. Each machine storage device a HDD and one SSD. HDFS used the following options: (1) 256MB block size, (2) ONE_SSD mode, (3) block replication factor is 3. Tajo settings are default. For the Performance evaluation, we used Select count(*) query of SQL language and TPCH benchmark.

5.2 Select Count (*) Performance

Select count query is intended to focus on storage I/O performance. The select count(*) query counts records from a table. To count records number, all data in the table should be read. Therefore, we can see intuitively the storage I/O performance. We used lineitem table which has 74.12GB data. We performed experiment five times and calculated the average. Figure 5 shows select count(*) query execution times.

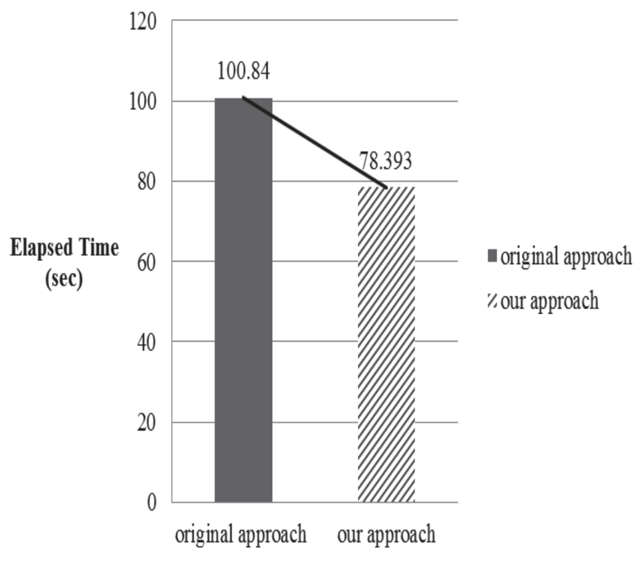

Figure 5: Total Elapsed time of select count(*) from lineitem

From the above results and analysis, we demonstrated that the elapsed time of our approach was shorter than the original approach. On average, the query execution time was improved up to 22%. This is because our approach read almost data from SSDs. Sometimes, it read block from the HDD in the local DataNode or the SSD in the same rack DataNode. However, an observation is that almost case read the block from the SSD in the local DataNode.

5.3 TPC-H Performance

We used the Transaction Processing Performance Council (TPC) benchmark H (TPC-H). TPC-H benchmark is decision support benchmark. It use large amounts of data, execute queries, and return answers. By performing the TPC-H benchmark, we can analysis the overall cost of Hadoop system, including CPU and storage I/O costs. Some of TPCH queries are available in the current Tajo. Six queries which are available in Tajo as follows.

Table 1: TPC-H query

Num	TPC-H Query
Q1	**select** l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extendedprice*(1-l_discount)) as sum_disc_price, sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge, avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order from lineitem where l_shipdate <= '1998-09-01' **group by** l_returnflag, l_linestatus order by l_returnflag, l_linestatus
Q2	**select** s_acctbal, s_name, n_name, r2_1.p_partkey, p_mfgr, s_address, s_phone, s_comment from r2_1 join r2_2 on r2_1.p_partkey = r2_2.p_partkey **where** ps_supplycost = min_ps_supplycost **order by** s_acctbal, n_name, s_name, r2_1.p_partkey
Q3	**select** l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate, o_shippriority from customer as c **join** orders as o on c.c_mktsegment = 'BUILDING' and c.c_custkey = o.o_custkey **join** lineitem as l on l.l_orderkey = o.o_orderkey **where** o_orderdate < '1995-03-15' and l_shipdate > '1995-03-15' **group by** l_orderkey, o_orderdate, o_shippriority **order by** revenue desc, o_orderdate
Q6	**select** sum(l_extendedprice*l_discount) as revenue from lineitem **where** l_shipdate >= '1994-01-01' and l_shipdate < '1995-01-01' and l_discount >= 0.05 and l_discount <= 0.07 and l_quantity < 24;
Q12	**select** l_shipmode, sum(case when o_orderpriority ='1-URGENT' or o_orderpriority ='2-HIGH' then 1 else 0 end) as high_line_count, sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH' then 1 else 0 end) as low_line_count from orders, lineitem **where** o_orderkey = l_orderkey and (l_shipmode = 'MAIL' or l_shipmode = 'SHIP') and l_commitdate < l_receiptdate and l_shipdate < l_commitdate and l_receiptdate >= '1994-01-01' and l_receiptdate < '1995-01-01' **group by** l_shipmode **order by** l_shipmode
Q14	**select** 100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice*(1-l_discount) else 0 end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue from lineitem, part **where** l_partkey = p_partkey and l_shipdate >= '1995-09-01' and l_shipdate < '1995-10-01'

In the experiments, we measured the elapsed time and compared the execution time of each queries. We performed it five times and calculated the average. Figure 6 shows measurement results.

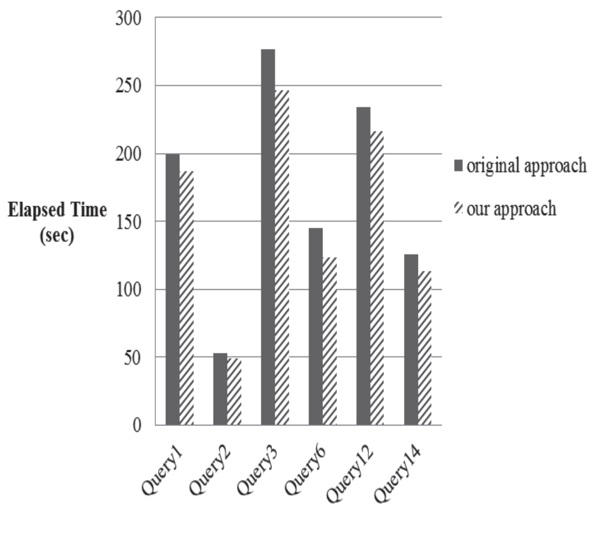

Figure 6: Total elapsed time of TPC-H queries

When using our approach, the average elapsed time of six queries was also reduced. Experiment results showed that the overall Hadoop system performance improved from 4% up to 30% on average.

This is because most of the blocks were read from the SSDs based on our approach. Improved the HDFS storage I/O throughput reduces the overall execution time of TPCH benchmark effectively.

6. CONCLUSION

The performance of a Big-data system is largely influenced by its CPU, memory, and storage I/O. No matter how high the performance of the CPU or Memory, if the storage I/O performs slowly, it is difficult to improve the overall performance of a Big-data system. Therefore, improving the performance of storage I/O is very important.

In this paper, we propose a hybrid block-selection method on Hadoop Eco System based on storage type. Our approach differs from the original method because it considers the storage type. When a read request is submitted by a client, the distributed processing system examines the storage type and selects the block in the SSD of DataNode among different candidate DataNodes. The block in the SSD is read first. By using the high SSD bandwidth first, the HDFS storage I/O is improved, as demonstrated by our experiment. In the experiments, our method efficiently reduced the execution time of select count(*) query and TPCH benchmark up to 22% and 30% respectively. As a result, we have demonstrated that our approach contributes to the improvement of the overall performance of Hadoop Eco System. We expect that SSDs will be more cost-effective in the future. Accordingly, our new block-selection methods will take more SSDs to you by employing inexpensive and high-performance SSDs.

7. ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (NRF-2015R1A2A1A05001845).

8. REFERENCES

[1] Ahn, S. Y., Park, S. K., Hone, J. K. and Chang, W. S. Performance Implications of SSDs in Virtualized Hadoop Clusters. Big Data (BigData Congress) IEEE. (2014): 586 - 593.

[2] Choi, H., Son, J., Yang, H., Ryu, H., Lim, B., Kim, S., and Chung, Y. D. Tajo: A Distributed Data Warehouse System on Large Clusters. Data Engineering (ICDE). IEEE. (2013): 1320–1323.

[3] Floratou, A., Minhas, U. F., and Ozcan, F. SQL-on-Hadoop: Full circle back to shared-nothing database architectures. Architectures, Journal Proceedings of the VLDB, 7.21 (2014): 1295–1306.

[4] Honjo, T., Oikawa. K, Hardware acceleration of Hadoop MapReduce. Big Data IEEE. (2013): 118 - 124

[5] Jeon, H., Maghraoui, K. E., Kandiraju, G. B. Investigating Hybrid SSD FTL Schemes for Hadoop Workloads. International Conference on Computing Frontiers. (2013).

[6] Kambatla, K. and Chen, Y. The Truth About MapReduce Performance on SSDs. Large Installation System Administration Conference (LISA), 2014. USENIX Association.

[7] Kang, S. H., Koo, D. H., Kang, W. H., and Lee, S. W. A Case for Flash Memory SSD in Hadoop Applications.

[8] Krish, K. R., Anwar, A., Butt, A. R. hatS: A Heterogeneity-Aware Tiered Storage for Hadoop. Cluster, Cloud and Grid Computing (CCGrid) IEEE. (2014): 502 - 511.

[9] Leu, J. S., Yee, Y. S., Chen, W. L. Comparison of Map–Reduce and SQL on largescale data processing. Parallel and Distributed Processing with Applications (ISPA). (2010): 244 – 248.

[10] Moon, S., Lee, J., and Kee, Y. S. Introducing SSDs to the Hadoop MapReduce Framework. Cloud Computing (CLOUD) IEEE. (2014): 272–279.

[11] Özcan, F., Tatbul, N., Daniel, J. A., Kornacker, M., Mohan, C., Ramasamy, K., Wiener, J. Are We Experiencing a Big Data Bubble?. SIGMOD International conference on Management of data. (2014):1407-1408.

[12] Saxena, P. and Chou, D. J. How Much Solid State Drive Can Improve the Performance of Hadoop Cluster? Performance evaluation of Hadoop on SSD and HDD. International Journal of Modern Communication Technologies & Research (IJMCTR), (2014).

[13] Saxena, P. Kumar, P. Performance evaluation of HDD and SSD on 10GigE, IPoIB & RDMA-IB with Hadoop Cluster Performance Benchmarking System, Confluence The Next Generation Information Technology Summit (Confluence). (2014):30 – 35.

[14] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The Hadoop Distributed File System. Mass Storage Systems and Technologies (MSST), IEEE. (2010): 1–10.

[15] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. Hive: A Warehousing Solution Over a Map-Reduce Framework. Journal Proceedings of the VLDB, 2.2 (2009): 1626–1629.

[16] Wei, Tan., Fong, L., Yanbin, Liu. Effectiveness Assessment of Solid-State Drive used in Big Data Services. Web Services (ICWS) IEEE. (2014): 393 - 400.

[17] Wu, D., Xie, W., Ji, X., Luo, W., He, J., Wu, D. Understanding the Impacts of Solid-State Storage on the Hadoop Performance. Advanced Cloud and Big Data (CBD). (2013):125 – 130.

[18] Xu, Y., Kostamaa, P., Gao, L. Integrating Hadoop and Parallel DBMS. SIGMOD International Conference on Management of data. (2010):969-974.

[19] Yang, Q. and Ren, J. I-CASH: Intelligently Coupled Array of SSD and HDD. High Performance Computer Architecture (HPCA) IEEE. (2011): 278–289.